Resilience through Self-Configuration in the Future Massively Defective Nanochips

Piotr Zając (a,b), Jacques Henri Collet (a), Jean Arlat (a), and Yves Crouzet (a)

(a) LAAS- CNRS, Université de Toulouse,
7 av. du Colonel Roche 31077 Toulouse, France

(b) Department of Microelectronics and Computer Science,
Technical University of Lodz, Al. Politechniki 11, 93-590 Lodz, Poland

{pzajac, jacques.collet, jean.arlat, yves.crouzet}@laas.fr

Abstract

This paper addresses the resilience challenges in the future nanochips made up of massively defective nanoelements and organized in a replicative multicore architecture. The main idea is to suggest that the chip should work with almost no external testing or control mechanisms, using a self-configuration methodology to ensure the resilience of operation even in the presence of a significant fraction of defective cores. By self–configuration, we mean self-diagnosis with mutual tests, self-shutdown of inactive cores and self-configuration of communications. We study the efficiency of the proposed methodology and show that the method is applicable up to a fraction of 30% of defective cores in the on-chip network.

1 Introduction

Thanks to size reduction and technology advances, chips with an extremely large number of transistors, say typically several hundreds billions transistors, will become feasible in the next decade. While this trend, together with accelerated clock speed, is prone to result in substantial performance improvements for future processors, some real challenges are to be faced to achieve actual improvements [1]. These challenges include: defects and variability of the elementary devices as well as design complexity and communication bandwidth limitation. In this context, the motivation of the work presented in this paper results from two main observations:

1) Future technologies will make it increasingly more difficult to reach suitable yield levels without relying on fault tolerance techniques.
2) In the design of current processors, performance increase is based on the multiplication of the number of cores.

Extreme downsizing inevitably results in atomic range dimensions, thus, in inter- and intra-device variability [2,3] and ultimately in massively defective technologies, thus impairing the production yield of such nanochips. To cope with this problem, much progress has been made for what concerns memory chips. Most advanced techniques consist in providing spare elements (lines, rows or words) in order to dynamically replace some defective elements. Indeed, techniques have been proposed that not only cope with production defects but also with faults occurring at runtime. Such techniques are primarily meant to achieve a high yield, which may require a significant overhead. For example, in [4], it is shown that for a 1Mb-chip and a cell defect ratio of 3%, a near 100% yield can be achieved, but at the cost of close to 100% overhead. Efficient fault tolerance techniques have been proposed also for processor chips. For example, the technique being recently proposed in [5] features a set of fragmented MPUs for which redundant fragments are available.

One may expect that, in the near future, the number of cores manufactured and the organization of processor chips will evolve significantly beyond the multicore symmetric multiprocessor (SMP) architectures such as today's popular bi, or quadricore processor chips. The recent announcement for a 80-core chip by Intel [6], already paves the way forward. Moreover, the consequence of this trend is that software techniques are evolving so as to take advantage of such multicore processor architectures. This means also that alternative resilience solutions can be readily considered and fully exploited. The goal goes beyond simply avoiding the delivery of defective chips (i.e., chips with defective cores) even by using spare elements at production time. Alone, such an approach would require increasingly — perhaps prohibitively — high effort and cost in manufacturing and testing. We advocate a more pragmatic approach that is to maximize the capacity to exploit the valid cores available on a chip. The underlying rationale being that at the end of the manufacturing process, the chips would then be sorted according the achieved MIPS performance level, as is usually the case with respect to clock frequency. Consequently, instead of sorting the chips according to their frequency (1.6 Ghz, 1.8 Ghz or 2 Ghz), one may think of sorting them as a function of the number of valid

cores. Such a principle is already applied on the production lines of some manufacturers (e.g., Intel Core Duo chips featuring a defective device are "recycled" as Core Solo chips[1]).

The approach we are presenting is somewhat more ambitious, as we are also considering applying such a self-configuration dynamically at runtime rather than simply statically at production time via an external diagnosis. This way, it would be possible to continue using the processing resources available onchip even when faults occurring in operation will impair some additional cores. The performance would be simply gradually reduced accordingly (as aging process in real life!). We consider general purpose network (GPN) chips based on a 2D-mesh architecture as target for our study. We also identify the processing core to be the basic unit for reconfiguration. The underlying idea is to be able to discriminate between valid and defective cores. In practice, it would be unrealistic to consider diagnosing all nodes and routes in such a kind of very complex chip via some external equipment. Accordingly, the diagnosis should involve as little external interventions as possible (e.g., to control the start up phase and the subsequent operation). Our contribution consists in suggesting and studying an autonomous reconfiguration methodology that involves:

a) Self-diagnosis of cores with cross-node tests.
b) Self-configuration of communication routes.
c) Self-shutdown of inactive cores.
d) Self-adaptative redundancy management at runtime.

Steps a-c must be executed at startup, and possibly periodically at runtime. Step d is executed at runtime exclusively. In this paper, we focus on self-configuration at startup (i.e., steps a-c). The expected benefits from such self-configuring capability are twofold:

1) Achieving resilient operation in GPNs, despite the presence of a fraction of defective cores.
2) Enabling a smooth degradation of the chip performance as a function of the number of defective cores at runtime.

The paper is organized as follows: In Section 2, we briefly introduce the features characterizing the type of GPN architecture we are considering that are relevant for our study. Section 3 describes how to conduct self-diagnosis through mutual tests; in particular, we show that mutual tests enable splitting the chip into zones of consistent cores such that all cores are good or all are defective in a zone. In Section 4, we study the efficiency of the routing discovery to assess how many valid cores can be reached via a flooding protocol as a function of the node failure probability and of the chip size. Section 5 describes the principle adopted to self-shutdown the cores, which cannot take part into the processing. Finally, Section 6 provides some concluding remarks.

[1] http://en.wikipedia.org/wiki/Intel_Core.

2 The architectural framework

The typical kind of architectures that we are considering consists in general-purpose chips based on a 2D-mesh architecture. Figure 1 depicts an example of such an architectural framework with 9x5 nodes and one single input/output port (IOP).

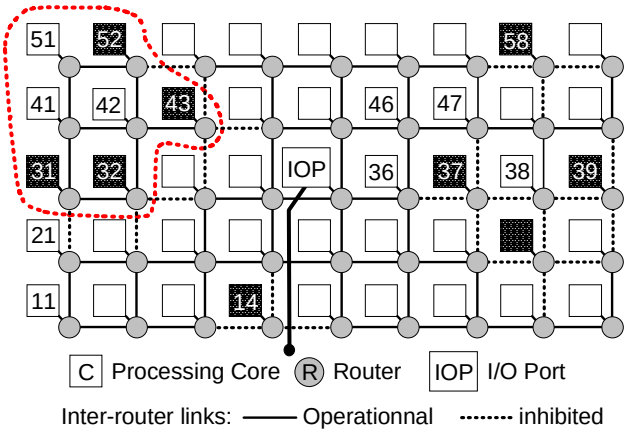

Figure 1: Example of a 9x5 node network architecture

Each node is made up of a processing core (C) associated to a router (R), which forwards the incoming messages to enable intercore communications. Lines connecting the routers materialize inter-router links. In this figure, blackened cores depicts defective ones; 9 cores are identified as such: C_{52}, C_{58}, C_{43}, C_{31}, C_{32}, etc. This would correspond to a massively defective chip featuring a failure core probability of about 20%. Also, dashed lines identify logically disconnected links as explained in Section 3. The zone enclosed in a dashed line in the top left corner of the figure shows that a cluster of 7 cores is unavailable due to the failure of cores C_{52}, C_{43}, C_{32}, C_{31}.

In this study, we concentrate on core failures. The core failure probability (P_F) is an adjustable parameter in our study that will be varied typically from 1% to 20-30%. The consideration of link failures and of appropriate mechanisms to cope with them (e.g., see [7]) is not addressed in this paper. We also consider that each core is equipped with a non-volatile memory (e.g., a flash memory not shown in Figure 1) that stores test vectors and several bits to save the results of the previous diagnoses even when the power supply is turned off (see Section 3).

The problem is that, at startup (i.e., when the chip is powered on), it is not known which cores are defective and which communication routes are available to exchange messages between nodes, especially between the IOP (whose role is to allocate tasks) and the valid idle nodes.

3 Self-diagnosis via mutual testing

The simplest method to identify defective cores is core self-test. In this approach, every core executes its test instructions and compares the outputs that it generates with the result vectors also stored in the non-volatile memory. Unfortunately, a faulty core can behave unpredictably, and diagnoses itself as non-faulty, *independently* of the state of its neighbors. Thus, mutual diagnosis [8] is much preferred. In this alternative approach, every core (say A) in the network executes the diagnosis program simultaneously to each neighbor (say B) and both compare the generated results. If the results differ and if A is good (i.e., fault-free), it concludes that B is faulty. If A is faulty, its response is unpredictable and it can diagnose B as good or faulty. In Table 1, the two leftmost columns display the four possible real states of two adjacent cores and the two rightmost columns reveal the corresponding mutual diagnosis. For sake of concision GOOD and FAULTY will be denoted as G and F, respectively, and X means that the result of the diagnosis is unpredictable (i.e., either G or F).

Actual state of core A	**Actual state of core B**	Diagnosis for core A made by core B	Diagnosis for core B made by core A
GOOD	GOOD	GOOD	GOOD
GOOD	FAULTY	X	FAULTY
FAULTY	GOOD	FAULTY	X
FAULTY	FAULTY	X	X

X means undetermined diagnosis, possibly GOOD or FAULTY.

Table 1: Diagnoses using mutual testing

It turns out that the diagnosis G-G may correspond to any combination of real states in the self-test approach (namely G-G, G-F, F-G or F-F) whereas, in the mutual test, the sole possible real states are G-G or F-F.

Mutual diagnosis may be used to split *de facto* the core array is several zones through the following isolation mechanism: when a G-core (i.e., a core in a G state) concludes that a neighbor is faulty, it stores a Boolean variable in its memory (or possibly in the router memory) to inhibit all communications with this neighbor. In other words, the G-core executes a logical inhibition (not a physical disconnection) of the F-core. For instance, in Figure 1, C_{36} will conclude that C_{37} is faulty and it will disable all communications with this core. The link between C_{36} and C_{37} therefore appears as dashed. This allows G-cores to build up one or several simple-connected zones (SCZ) in the network enabling full propagation of messages between any two cores inside each of these zones. The border of a SCZ is made up with F-cores (diagnosed as faulty by neighbor G-cores), and consequently logically disconnected. In this approach, the SCZ enclosing the IOP is especially important as all cores outside this zone are lost for the processing. For sake of clarity, let us refer again to Figure 1. The dotted zone in the top left corner shows a cluster of cores disconnected from the main SCZ, which are lost for processing. The issue is that some cores in this cluster are in the good state but lost, see cores C_{41}, C_{51} and C_{42}. This will become especially problematical when P_F increases.

As we previously stressed, all cores in a SCZ are in the same state (i.e., all cores are good or all are faulty). This property holds in particular for the SCZ enclosing the IOP, which includes the cores that will execute the processing. Unfortunately, there is one pernicious issue in that we don't know the real state of the cores in this SCZ, except that the probability to get a SCZ with n G-cores or n F-cores is $\approx (1 - P_F)^n$ or $\approx P_F^n$, respectively. To remove this ambiguity, we suggest executing an external diagnosis of the *sole* IOP to make sure that it works correctly. Ultimately, self-diagnosis is not completely autonomous and requires a minimal external test for chip validation.

4 Self-configuration of communications

The step that we consider here consists in discovering the routes connecting the IOP to G-cores. Remember that valid routes are unknown, because the topology of the SCZ including the IOP is *a priori* unknown and because it might be necessary to move around (clusters of) defective nodes within the SCZ. Nevertheless, these routes are crucial for chip operation, as the IOP will use them to allocate incoming tasks to idle cores. The basic idea to execute task allocation at runtime is that the IOP should include a *buffer of valid routes* (VRB) to the idle cores, so that the IOP allocates incoming processes by searching a core in this buffer. Of course, this buffer is initially empty. Consequently, route discovery must be executed at startup to initialize the VRB. We describe in Section 4.1 the route-discovery protocol. The efficiency of this protocol is then studied in Section 4.2.

4.1 Contract net protocol

Route discovery is achieved by means of a *contract net protocol* (CNP) based on message multicast [9], which can be typically decomposed in the two phases (request emission and acknowledgment) that we describe hereafter.

4.1.1. Request phase: First, the IOP emits a *request message* (RM). We consider broadcast diffusion, where each node forwards each incoming RM by copying it on all links except the incoming link [10]. Broadcast protocols are successful in discovering routes (with high probability as will be shown in section 4.2) that move around disconnected zones, i.e., around zones including cores diagnosed as F-cores in the mutual-test procedure depicted in Section 2.

4.1.2. Acknowledgment phase: Each valid node receiving the RM sends an *acknowledgment message* (AM) towards the IOP. Globally, the number of AMs returning to the IOP is as large as the number of contacted nodes in the SCZ enclosing the IOP. However, the AM is not broadcast. We

suggest rather that each time a node forwards the RM in phase 1 of the protocol, it adds in the RM route field the input and output link IDs. For instance, for a scheme with 4 links per node as in Figure 1, one may consider the following coding ID_{North}=00, ID_{East}=01, ID_{South}=10, and ID_{West}=11. A node adding 0111 to the route field of the request message tells that the message came in through East and was forwarded through West. Note also that no absolute node addressing is needed in this communication approach. In this context, the AM in phase 2 simply follows the RM route in the opposite direction, which dramatically limits the number of reemissions compared to the broadcast mechanism implemented in phase 1.

For clarity, we illustrate in Figure 2 the mutual test and the resulting limited propagation of the RM considering a 1-D network with 8 cores connected to the IOP. Cores 1, 6 and 8 are defective.

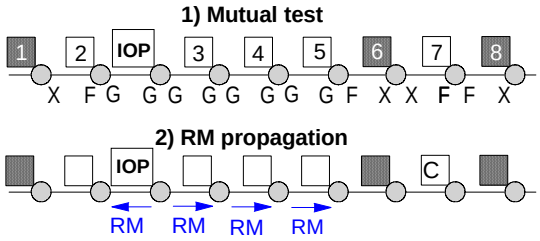

Figure 2: Illustration of mutual test and RM propagation

Figure 2.1 shows the mutual diagnosis step. When the IOP tests its neighbors (Cores 2 and 3), its diagnoses them as good, and this result is stored in the IOP router with setting the East bit and the West bit to G. When Core 2 tests its neighbors, its diagnoses Core 1 as faulty and the IOP as good, and this result is stored in its router by setting the West bit to F (for faulty) and East bit to G (for good), and so on, etc. After the diagnosis has been executed by all cores, we get the final result as: X FG GG GG GG GF XX FF X, Figure 2.2 shows the propagation of the RM across the network. Because of the logical disconnection mechanism described in Section 3, the propagation is blocked after Core 5 in East direction and after Core 2 in West direction.

4.2 Route discovery efficiency

We study in this paragraph the efficiency of the *request phase*, i.e., the capability to contact cores in a faulty network via the broadcast mechanism.

RM broadcasting was simulated using the multiagent simulator MASS[2]. Briefly, MASS is a Windows® application developed at LAAS, which calculates the temporal evolution of any system that can be described in terms of coexistence of state automata (SA). The full description of MASS is beyond the scope of this paper. MASS was successfully used in multiagent simulations, communication, prey-predator games. In the route discovery studies considered here, each node is represented by a SA, which forwards incoming messages. All nodes are activated in the asynchronous mode through a global scheduler. The principle of each simulation is as follows:

1) The simulator randomly generates a fraction NP_F of holes in the network. Holes represent the faulty cores logically inhibited following the mutual test process.
2) The IOP emits a RM, which is broadcast across the defective network, following the "hot potatoes forwarding" that was described in Section 4.1.
3) The program calculates the number of nodes *n* receiving the RM. The array *table[N_{MAX}]* is created and the entry *table[n]* is incremented every time exactly *n* nodes received the RM.
4) This procedure is repeated typically 2.000 times. At the end of the simulation, the entry *table[i]* determines the number of times *i* nodes were exactly reached. For instance, *table[26]*=110 would mean that the message reached exactly 26 nodes in 110 simulations. The probability to reach 26 nodes is therefore estimated as p(26)=110/2000.

Figure 3 shows the *Probability that the IOP reaches at least the fraction η of cores* in a network comprising 10x10 cores.

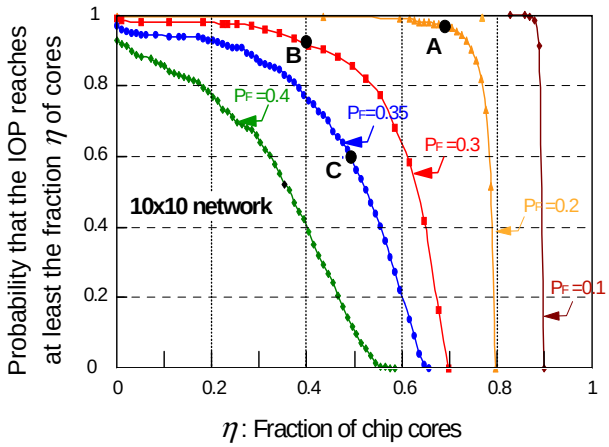

Figure 3: Probability for the IOP to reach a given fraction of cores (10X10 core network chip)

This figure provides a quantitative assessment of the (anticipated) fact that the more defective nodes (i.e., the higher the P_F), the smaller the SCZ including the IOP and therefore the more difficult building efficient multicore chips in the case of massively defective technologies. Maybe, the simplest way to explain this figure is by way of considering a specific example. For instance, we deduce from point A (X_A = 0.68 and Y_A = 0.96) that the probability is approximately Y_A= 0.96 that the IOP reaches at least η = 68% of all cores when the P_F = 0.2. Remember that (on average) in this case, it is impossible to reach more than η = 80% of the cores. Thus, the average fraction of lost cores (i.e., not in the IOP zone and

[2] A detailed and comprehensive documentation is available from http://www.laas.fr/~collet together with the possibility of downloading the simulator.

unavailable for processing) is approximately 80 – 68 = 12%. It can be seen that the number of lost cores increases quickly with P_F.

From point B, we conclude that the probability is approximately Y_B=0.92 that the IOP reaches at least η = 40% of all cores for P_F = 0.2. Thus, point B means that the fraction 70 – 40 = 30% of cores is lost to participate to the processing. Figure 3 shows that, when P_F = 0.2, the probability decreases to 0.6 that the IOP reaches at least η = 60% of all cores in the main SCZ. Figure 4 below shows the results obtained for a chip featuring 30x30 nodes.

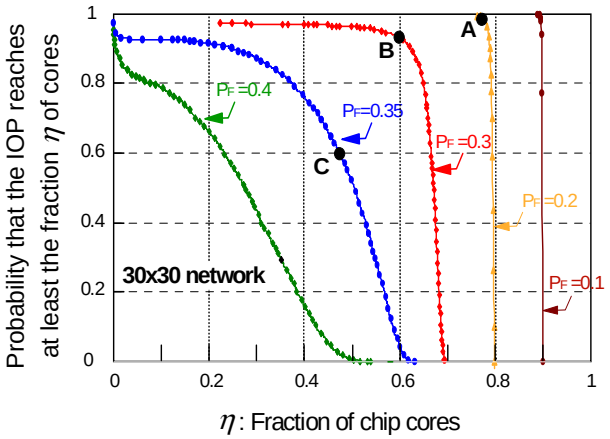

Figure 4: Probability for the IOP to reach a given fraction of chip cores (30X30 core network chip)

The comparison of these results with those in Figure 3 essentially shows that for $P_F \leq 0.35$, the probability that the IOP reaches a given fraction of nodes is higher in the network with 30x30 nodes than in the network with 10x10 nodes. For larger values of P_F, this is the opposite, as the risk is significantly increased of losing cores due to the fact they are enclosed into a cluster separated from the IOP SCZ by F-cores.

4.3 Production yield

It is interesting to correlate the production yield R to the probabilities $P(\eta, P_F)$ (displayed in Figures 2 and 3) that the IOP reaches a given fraction η of cores as a function of P_F. A chip will be validated if i) the IOP is operating (the corresponding probability is $1\text{-}P_F$) and ii) for instance at most one of its four direct neighbors is defective (the probability is $(1-P_F)^4 + 4P_F(1-P_F)^3$). The production yield in this example will be respectively $R(4) = (1-P_F)^5 P(\eta, P_F)$ (for 4 IOP neighbor G-cores) and $R(3) = 4P_F(1-P_F)^4 P(\eta, P_F)$ (for 3 IOP neighbor G-cores out of 4). Figure 5 displays the estimation of the production yield as $R(4)+R(3)$ in a 10x10-core chip for various values of P_F.

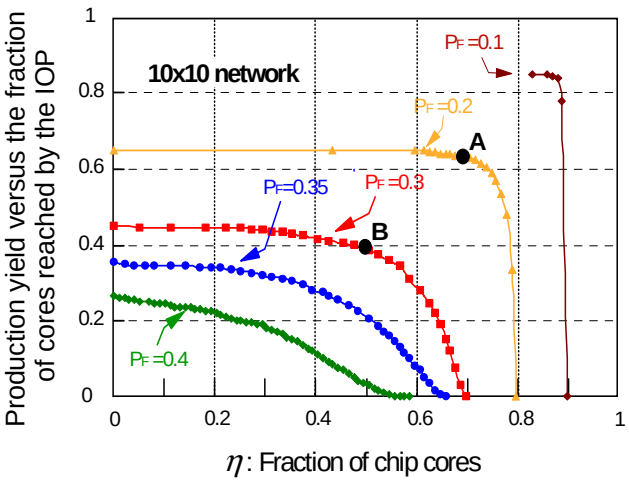

Figure 5: Estimation of the production yield

Point A means that, when P_F = 0.2, the production yield achieved by validating chips in which the IOP may reach at least 70% of the cores is about 0.65. Note that the reduction of the production yield becomes quickly catastrophic when considering P_F > 0.2. For instance, point B means that, when P_F = 0.3, the production yield achieved by validating chips in which the IOP may contact at least 50% of the cores is about 0.4. In that case, not only the production yield is very low, but moreover, half of the cores may be lost for processing!

5 Self-shutdown of cores

In this section, we consider limiting the power consumption of chips by shutting down all cores *outside* the SCZ enclosing the IOP. Remember that these cores are lost for processing, and that in the massively defective technologies they may represent a significant core fraction. We illustrate the proposed self-shutdown method considering the 1-D network already depicted in Figure 2. Indeed, Figure 6 may be viewed as the temporal continuation of steps 1 and 2 in Figure 2.

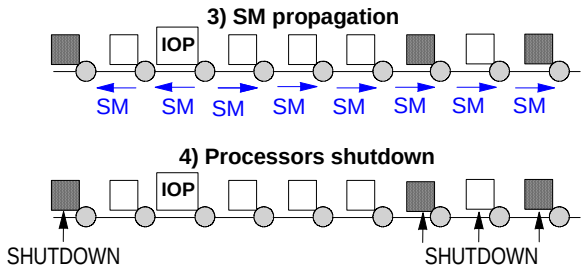

Figure 6: Illustration of the self-shutdown method

In step 3 (Figure 6.3), we consider that a second message (say the *shutdown message* - SM) is emitted by the IOP and again broadcast across the network. Contrarily to the RM, it is forwarded by each router independently of the diagnosis results deduced from the mutual test. Thus, this message is

not blocked at the border of the SCZ and it may propagate across the "faulty zones" which cannot take part to the processing. Now, let us reasonably assume that routers are entities much smaller than cores, so that the probability of router failure is much lower than that of cores. Consequently, there will be no (or almost no) defective routers in the network and, in particular in the faulty zones. Thus, the SM will propagate across the whole network. The final step consists in defining in the router automaton the action of shutting down the core when the router did not receive the RM, but the SM (Figure 6.4). Consequently, all (or almost all) cores which cannot participate to the processing, and which are outside the main SCZ including the IOP will be shutdown. This is particularly useful to reduce the power consumption in massively defective technologies.

Additionally, let us stress that the method described in Section 3 is based on the *complete* disconnection of each defective node, i.e., the disconnection of the core *and* of the router. This mechanism creates holes in the communication network, possibly resulting in unbalanced traffic, bottlenecks and the need for route discovery as described in Section 4. Now, we just showed that if routers are structurally simple and thus fault-free, it becomes possible to shutdown all cores outside the SCZ of the IOP (the router can keep track of this action by way of a dedicated private register) and moreover to keep using the routers in the faulty zones (outside the SCZ including the IOP) for broadcasting messages.

6 Conclusion

We have described a self-configuration methodology to tolerate defective nodes and to enable reliable operation of multicore chips in massively defective technologies. Self-configuration includes self-diagnosis of cores with mutual tests, self-configuration of communications and self-shutdown of inactive cores not in the SCZ enclosing the IOP. We studied the efficiency of the routing discovery mechanism in a 2D-mesh, i.e., how many valid cores can be reached by the IOP via a flooding protocol as a function of the node failure probability P_F and of the size of the chip. The results (shown in Figures 2 and 3) may be summarized as follows: When the $P_F > 0.2$, it becomes extremely difficult to efficiently produce chips that will feature at least 60% of valid cores and at most one defective core adjacent to the IOP.

Two additional comments can be added:

1) The results in Figures 2 and 3 ultimately define the upper core failure P_F (versus the fraction of cores) tolerable to achieve efficient processing or in other words, to warrant that a minimal fraction of cores will be able to participate to the processing controlled by the IOP. Consequently, in massively defective technologies, one must consider a compromise between the possible top complexity of the core and the necessary hardware overload to maintain P_F under the values deduced from this study. Thus, the more defective the technology, the more one will be forced to consider simpler cores, which poses the problem of the massive parallelization of applications.

2) The insights gained from this study rest on the assumption that the mutual test mechanism is perfect, i.e., that the fault coverage of test vectors is complete. It is well known that this is generally not true, especially if the core implements a complex micro-architecture. Thus, it will be safe to consider that any core deemed as fault-free could be ultimately faulty, especially in massively defective technologies, when P_F is as high as 0.2 or 0.3. Consequently, a safe and conservative position will be to consider another fault-tolerance layer at runtime, based on the redundant execution of the applications among the available nodes. Additionally, this fault tolerance layer will enable for coping with transient faults.

References

[1] R. I. Bahar, D. Hammerstrom, J. Harlow, W. H. Joyner, C. Lau, D. Marculescu, A. Orailoglu, M. Pedram, "Architectures for Silicon Nanoelectronics and Beyond," *Computer*, vol. 40, no. 1, pp. 25-33, January 2007.

[2] A. J. Bhavnagarwala, X. Tang, J. D. Meindl, "The Impact of Intrinsic Device Fluctuations on CMOS SRAM Cell Stability," *IEEE Journal on Solid-State Circuits*, vol. 36, no. 4, pp. 658–665, April 2001.

[3] S. Roy, A. Aaenov, "Intrinsic Parameter Fluctuations in Nano-scale CMOS," *Science*, vol. 309, no. 5733, pp. 388-390, July 2005.

[4] M. Nicolaïdis, N. Achouri, L. Anghel, "A Diversified Memory Built-In Self-Repair Approach for Nanotechnologies," in *Proc. 22nd IEEE VLSI Test Symp (VTS'2004),* Napa Valley, CA, USA, 2004, pp. 313-318, (IEEE CS Press).

[5] T. Nakura, K. Nose, M. Mizuno, "Fine-Grain Redundant Logic Using Defect-Prediction Flip-Flops," in *Proc. IEEE International Solid-State Circuits Conference (ISSCC-2007),* San Francisco, CA, USA, 2007, (IEEE CS Press).

[6] S. Vangal *et al.*, "An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS " in *Proc. IEEE International Solid-State Circuits Conference (ISSCC-2007),* San Francisco, CA, USA, 2007, (IEEE CS Press).

[7] S. Murali, T. Theocharides, N. Vijaykrishnan, M. J. Irwin, L. Benini, G. D. Micheli, "Analysis of Error Recovery Schemes for Networks on Chips," *IEEE Design & Test of Computers,* vol. 22, no. 5, pp. 434- 442, Sept.-Oct. 2005.

[8] L.E. Laforge, K. Huang, V.K. Agarwal, "Almost Sure Diagnosis of Almost Every Good Elements," *IEEE Trans. on Computers*, vol. 43, no. 3, pp295-305, 1994.

[9] R.G. Smith, "The Contract Net Protocol: High-level Communication and Control in a Distributed Problem Solver," *IEEE Trans. on Computers*, vol. 29, pp. 1104-1113, 1980.

[10] Y.K. Dalal, and R.M. Metcalfe, "Reverse Path Forwarding of Broadcast Packets," *Communications of the ACM*, vol. 21, no. 12, pp.1040-1048, 1978.